ARTIFICIAL INTELLIGENCE FOR MANUFACTURING
AI61009
TERM PROJECT
on
Machining Operation Sequence Optimization

By

Anwesha Patel (20ME10020)
Ashutosh Dash (20ME3FP32)
Abhishek Ranjan(20ME10004)

Under the Supervision of
Prof. Sankha Deb

Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Contents

Introduction

Optimizing the sequence of operations is important factor in improving the productivity of machines and reducing the production cost. Process planning is an essential component for linking design and downstream manufacturing processes. Given CAD data of a part, the goal of a computer-aided process planning (CAPP) system is to generate a series of manufacturing instructions for the part. In operation sequencing, it is necessary to apply good manufacturing practices and maintain the consistency of the desired functional specifications of a part. A good sequence of operations can ensure low machining cost (caused by machine utilization, setups, tool changes, etc.) and satisfy precedence constraints among the operations. However, finding an optimum operation sequence with low machining costs in a huge search space bound by production requirements is an insurmountable challenge**[1].**

One approach in optimization is straightforward and requires considerable computation power: brute force methods which try to calculate all possible solutions and decide afterwards which one is the best. These methods are feasible only for small problems (in terms of the dimensionality of the phase space), since the number of possible states of the system increases exponentially with the number of dimensions. In the case of continuous predictor variables, the number of states are infinite. Despite these drawbacks, brute force methods do have a few benefits: they are simple to implement, and in the case of discrete systems, all possible states are checked. As a consequence, brute force methods are often seen as reference methods for calculating the number of states, or the number of calculations necessary to find the optimum with a probability of 100%. Hence, it can be used for the estimation of the effort to solve a problem.**[11]**

Literature Review

For parts with complex structures and features, operations sequencing is well known as a complicated decision problem. The major difficulties include: (a) the search space is usually very large, and many previously developed methods could not find optimized solutions effectively and efficiently; (b) there are usually a number of precedence constraints in sequencing operations owing to manufacturing practice and rules, which make the search more difficult **[1]**.To address these issues, some optimization approaches based on modern heuristics or evolutionary algorithms, such as the genetic algorithm (GA) **[2–6]**, simulated annealing (SA) algorithm **[7,8]** and Tabu search algorithm **[8,9]**, have been developed in the last two decades, and significant improvements have been achieved. Most of the approaches usually stress on one or more objectives such as minimizing number of tool changes, setup changes, machine changes, and processing time, production cost. Brute-force approach calculates entire possible tool path pattern, and thereby calculates the optimized tool path pattern.

Problem Statement

This report aims at automatic generation of optimal sequence of machining operations in setup planning by Brute Force Approach based on minimizing the number of setup changes and tool changes, subject to various machining precedence constraints.

Input Parameters

For a given part, each of the machining features comprising the part is identified and assigned a unique serial number. Next the dimensions, tolerance, and surface finish of each feature are analyzed to determine the machining operation(s). Each operation is assigned a unique operation serial number and the operation type as per the convention given in Table 1. Figure 2 shows Tool Access Direction (TAD) convention used. Thus if a feature to be machined has a TAD opposite to the outward normal to a face, then that face number would be assigned as the TAD identifier for machining the feature. Next, the TAD corresponding to the machining operation is identified by the face plane number of the corresponding feature to be machined.**[10]**

Table 1. Operation type identifier.

Operation code	Operation type	Operation code	Operation type
1	Rough face milling	7	Reaming
2	Finish face milling	8	Boring
3	Rough end milling	9	Tapping
4	Finish end milling	10	Grooving
5	Centre drilling	11	Chamfering
6	Drilling	12	Filleting

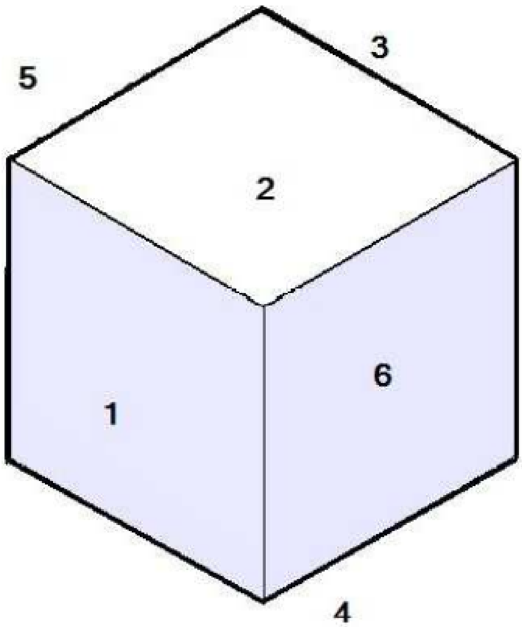

Fig. 2. TAD convention.

Kumar, Chandan, and Sankha Deb. "Generation of optimal sequence of machining operations in setup planning by genetic algorithms." *Journal of Advanced Manufacturing Systems* 11.01 (2012): 67-80.

Table 4. Input data.

Operation no.	Operation type	Feature no.	TAD
1	Face Milling	1	6
2	Face Milling	2	3
3	Face Milling	3	5
4	Face Milling	4	1
5	Face Milling	5	6
6	Face Milling	6	3
7	Face Milling	7	5
8	Face Milling	8	1
9	Face Milling	9	2
10	Face Milling	10	2
11	Face Milling	11	4
12	End Milling	12	1
13	End Milling	13	1
14	Drilling	14	2
15	Drilling	14	2
16	Drilling	14	2
17	Drilling	14	2
18	End Milling	15	3
19	End Milling	16	6
20	End Milling	17	5
21	End Milling	18	2
22	Boring	18	2
23	Drilling	18	2
24	Boring	19	2
25	Drilling	20	4
26	Boring	20	4
27	Drilling	21	2
28	Tapping	21	2

Kumar, Chandan, and Sankha Deb. "Generation of optimal sequence of machining operations in setup planning by genetic algorithms." *Journal of Advanced Manufacturing Systems* 11.01 (2012): 67-80.

A change in TAD implies a change in the setup. There are various operation precedence constraints that cannot be violated e.g. internal precedence constraints for machining of features, datum precedence constraints, parent–child precedence constraints, primary–secondary precedence constraints, etc.

Table 5. Precedence constraints data.

Precedence between operations	Description of the constraint
11→All	Datum and support face of part
9→24, 9→27, 9→28	Datum surface for operation
25→26, 10→9, 10→14, 10→15, 10→16, 10→17, 5→10, 6→10, 7→10, 8→10, 19→10, 20→10	Material removal interaction
2→6, 6→18, 4→12, 4→13, 4→8, 3→7, 20→7, 1→5, 19→5, 9→21	Datum surface for operation
21→23, 23→22, 25→24, 27→28	Fixed order of machining
26→9, 26→10, 18→4	Fixturing interaction

Formulation of Objective Function:

When multiple objective functions are combined into one single overall objective function, the weighted sums method is often used to construct the fitness function. For the present problem, the overall objective is to optimize the operation sequence by combining two objective functions, namely, minimizing the number of setup changes and minimizing the number of tool changes. Accordingly, the following rules for evaluating the fitness function for optimizing an operation sequence has been formulated **[10]**:

Case 1: If TAD changes, but operation remains same: number of setups is increased by 1.

Case 2: If both TAD and the operation change: number of setups is increased by 1; number of tool changes is increased by 1.

Case 3: If TAD does not change, but operation changes: number of tool changes is increased by 1.

Case 4: If both TAD indicator and operation remains same: number of tool changes remain unchanged.

<u>METHODOLOGY:</u>

For implementing sequence optimization baked on minimization of number of tool changes and setup changes, while keeping the precedence constraints intact, we have incorporated Brute Force algorithm that verifies and reports the best sequence out of 10,000 random sequences checked amongst all sequences possible.

Explanation of the algorithm implemented:

Before implementation of Brute Force algorithm, we iterated on our input list (numbers from 1 to 28, which represent our respective processes that need to be implemented on our machine part) in order to obtain 10,000 random permutations of our list. All of these permutations and lists were stored in another 2-dimensional array that requires to be checked in order to obtain our desired sequence. These sequences were then checked thoroughly according to the requirements of the problem statement and the best possible sequence was obtained, which varied each time as the selection of sequences were quite random and the choice was from an extremely large universal set containing 28 factorial elements.

Precedence Constraints and Matrix Design:

The given precedence constraints were first listed down in a csv file and then a precedence constraint matrix was designed. The ideology behind the matrix was that if the element in the ith row and jth column was 1, then the ith operation was allowed to be performed before the jth operation. While of the element was 0, then the reverse happens (ith operation cannot be performed unless jth operation has already been executed).

An image of our designed precedence matrix has been shown below:

```
0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,1
0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0
1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0
0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Control Flow of the Algorithm and usage:

The program starts by importing the csv file that contains the precedence matrix in a 2-Dimensional Array. Then we define a boolean function check_precedence_criteria to check if any given sequence satisfies our precedence matrix or not. Then we define 2 functions in order to calculate the number of tool changes and the number of orientation changes corresponding to an obtained sequence, namely calc_tool_changes and calc_orientation_changes.

Observable Loopholes in using Brute Force:

Computationally this process is not very effective as there might be cases where there may not be any sequence satisfying all our constraints among the chosen random set of 10,000 instances. Moreover there is a lot of comparison included while processing sequences by brute force hence it's time complexity factor is not quite preferable.

RESULTS:

Approach (code-based explanation):

The precedence matrix in a 2-Dimensional Array is imported into the program at the beginning. Then, to determine whether a particular sequence satisfies our precedence matrix, we define the boolean function check_precedence_criteria. Then, we construct two functions, calc_tool_changes and calc_orientation_changes, to calculate the number of tool changes and the number of orientation changes according to an achieved sequence.

```
def calc_orientation_changes(sequence):
    orien_changes = 0
    current_orientation = orientations[sequence[0]]
    for part in sequence[1:]:
        if orientations[part] != current_orientation:
            orien_changes += 1
            current_orientation = orientations[part]
    return orien_changes

def calc_tool_changes(sequence):
    tool_changes = 0
    current_tool = tools_grippers[sequence[0]]
    for part in sequence[1:]:
        if tools_grippers[part] != current_tool:
            tool_changes += 1
            current_tool = tools_grippers[part]
    return tool_changes
def fun(orien_changes, tool_changes, feasibility_index):
    output=1/(Wx*orien_changes + Wy*tool_changes - feasibility_index)
    return output
def CaculateFitness1(orien_changes, tool_changes, feasibility_index,fun):
    fitness = fun(orien_changes, tool_changes, feasibility_index)
    return fitness
```

Then, we build a second 2-Dimensional array with 10,000 random sequences drawn from a massive 28 factorial permutation range. After that, the control repeats each of these steps.

```
ct=0
rows,cols = (10000,28)
sequences = [[0]*cols]*rows
for i in itertools.permutations(parts):
    sequences[i][:]= list(i)
    ct=ct+1
    if(ct==10000):
      break
```

If any of these sequences satisfies the precedence constraints, they are first compared to another optimal sequence (if any are present) using a predetermined fitness function that gives the minimization of the parameters for the number of tool changes and the number of orientation changes priority. It is selected and kept as the new optimal if it occupies a better position than the previous one.

The control finally outputs the best answer found in the selected set after iterating through the 10,000 randomly selected samples. Finally, this is printed together with the relevant tool and orientation changes corresponding to our optimal sequence.

Printing each possible solution:

```
for sequence in sequences:
    if check_precedence_criteria(sequence) == True:
        prob_sequences.append(sequence)
        print ("Sequence: " + str(sequence))
        print ("Orientation changes: " + str(calc_orientation_changes(sequence)))
        print ("Tool gripper changes: " + str(calc_tool_changes(sequence)))
```

The above is a code sequence that prints each of the feasible solutions along with their number of tool and orientation changes.

Optimal solution comparison and printing:

```
if(check_precedence_criteria(sequence)==True):
  fi=0
  o1=calc_orientation_changes(sequence)
  t1=calc_tool_changes(sequence)
  out1=fun(o1,t1,fi)
  f1=CaculateFitness1(o1,t1,f1,out1)

  if(f_optimal<f1):
    optimal_sequence=sequence
    f_optimal=f1
```

This code segment primarily compares the fitness function values of the current state and the previous optimal states and takes a decision whether to change the optimal value to a new state or let it remain in the original state.

Solutions obtained as output:

The code was run several times and corresponding solutions obtained were subsequently saved. The 3 optimal sequences obtained post execution 4 times are represented in the following sequence of images:

```
Sequence:  [11, 2, 3, 1, 19, 5, 6, 20, 7, 25, 26, 8, 10, 14, 18, 4, 13, 12, 9, 21, 23, 22, 15, 16, 17, 24, 27, 28]

Orientation changes:   19

Tool gripper changes:  27
```

```
Sequence:  [11, 19, 1, 5, 2, 6, 3, 20, 7, 25, 26, 8, 10, 18, 4, 13, 12, 9, 21, 23, 22, 16, 15, 14, 17, 24, 27, 28]

Orientation changes:   16

Tool gripper changes:  27
```

```
Sequence:  [11, 2, 3, 19, 1, 5, 6, 20, 7, 25, 26, 8, 10, 18, 4, 13, 12, 9, 21, 23, 22, 16, 14, 15, 17, 24, 27, 28]

Orientation changes:   17

Tool gripper changes:  27
```

Hence the best possible optimal solution obtained of these 3 sequences is the second sequence and it correspondingly has 16 orientation changes and 27 tool gripper changes.

Conclusion

In this work, a brute force approach has been used for automatic generation of optimal, feasible operation sequence for machining prismatic parts based on minimizing the number of setups and tool changes subject to various precedence constraints. The resulting operation sequence has been found to satisfy most of the constraints imposed. The proposed methodology can be extended further by including other objective functions such as machining time and cost and new modifications can be made.

References

1. Guo, Y. W., et al. "Operation sequencing optimization using a particle swarm optimization approach." *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 220.12 (2006): 1945-1946.
2. Reddy, SV Bhaskara. "Operation sequencing in CAPP using genetic algorithms." *International Journal of Production Research* 37.5 (1999): 1063-1074.
3. Qiao, L., X-Y. Wang, and S-C. Wang. "A GA-based approach to machining operation sequencing for prismatic parts." *International Journal of Production Research* 38.14 (2000): 3283-3303.
4. Yip-Hoi, D. E. R. E. K., and DEBASISH DUTTA. "A genetic algorithm application for sequencing operations in process planning for parallel machining." *IIE transactions* 28.1 (1996): 55-68.
5. Zhang, F_, Y. F. Zhang, and Andrew Y. C. Nee. "Using genetic algorithms in process planning for job shop machining." *IEEE Transactions on evolutionary computation* 1.4 (1997): 278-289.
6. Ding, Lian, et al. "Global optimization of a feature-based process sequence using GA and ANN techniques." *International Journal of Production Research* 43.15 (2005): 3247-3272.
7. Ma, G. H., Y. F. Zhang, and A. Y. C. Nee. "A simulated annealing-based optimization algorithm for process planning." *International journal of production research* 38.12 (2000): 2671-2687.
8. Lee, Dong-Ho, D. Kiritsis, and P. Xirouchakis. "Search heuristics for operation sequencing in process planning." *International journal of production research* 39.16 (2001): 3771-3788.
9. Li, W. D., S. K. Ong, and A. Y. C. Nee. "Optimization of process plans using a constraint-based tabu search approach." *International Journal of Production Research* 42.10 (2004): 1955-1985.
10. Kumar, Chandan, and Sankha Deb. "Generation of optimal sequence of machining operations in setup planning by genetic algorithms." *Journal of Advanced Manufacturing Systems* 11.01 (2012): 67-80.

Websites:

11. http://www.statistics4u.com/fundstat_eng/cc_optim_meth_brutefrc.html